Available online at www.sciencedirect.com

ScienceDirect

Computational Geometry 37 (2007) 16–26

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Kinetic sorting and kinetic convex hulls

Mohammad Ali Abam [1], Mark de Berg [*,2]

Department of Computing Science, TU Eindhoven, PO Box 513, 5600 MB Eindhoven, The Netherlands

Received 9 May 2005; received in revised form 19 January 2006; accepted 8 February 2006

Available online 28 August 2006

Communicated by F. Hurtado and J.S.B. Mitchell

Abstract

Let S be a set of n points moving on the real line. The kinetic sorting problem is to maintain a data structure on the set S that makes it possible to quickly generate a sorted list of the points in S, at any given time. We prove tight lower bounds for this problem, which show the following: with a subquadratic maintenance cost one cannot obtain any significant speed-up on the time needed to generate the sorted list (compared to the trivial $O(n \log n)$ time), even for linear motions.

We also describe a kinetic data structure for so-called gift-wrapping queries on a set S of n moving points in the plane: given a point q and a line ℓ through q such that all points from S lie on the same side of ℓ, report which point $p_i \in S$ is hit first when ℓ is rotated around q. Our KDS allows a trade-off between the query time and the maintenance cost: for any Q with $1 \leqslant Q \leqslant n$, we can achieve $O(Q \log n)$ query time with a KDS that processes $O(n^{2+\epsilon}/Q^{1+1/\delta})$ events, where δ is the maximum degree of the polynomials describing the motions of the points. This allows us to reconstruct the convex hull quickly when the number of points on the convex hull is small. The structure also allows us to answer extreme-point queries (given a query direction $\vec{d}$, what is the point from S that is extreme in direction $\vec{d}$?) and convex-hull containment queries (given a query point q, is q inside the current convex hull?).

Keywords: Computational geometry; Kinetic data structures; Kinetic sorting; Lower bounds; Kinetic convex hulls

1. Introduction

Background. In many areas of computer science one has to store, analyze and manipulate geometric data. More and more often this involves objects in motion. Hence, the study of geometric data structures for moving objects has recently attracted a lot of attention in computational geometry, especially since Basch et al. [8] introduced the *kinetic-data-structure* (KDS, for short) framework [1,4,5,7,8,12–14].

A KDS is a structure that maintains a certain attribute of a set of continuously moving objects—the convex hull of moving objects, for instance, or the closest distance among moving objects. It consists of two parts: a combinatorial description of the attribute and a set of certificates with the property that as long as the outcomes of the certificates

* Corresponding author.
E-mail address: mdberg@win.tue.nl (M. de Berg).

[1] Supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 612.065.307.
[2] Supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2006.02.004

do not change, the attribute does not change. In other words, the set of certificates forms a proof that the current combinatorial description of the attribute is still correct. It is assumed that each object follows a known trajectory so that one can compute the failure time of each certificate. Whenever a certificate fails—we call this an *event*—the KDS must be updated. The KDS remains valid until the next event. The goal when designing a KDS is to make sure there are not too many events, while also ensuring that the update time at an event is small—see the survey by Guibas [11] for more background on KDSs and their analysis.

Computing the convex hull of a set of points in the plane is a classic problem in computational geometry. It is therefore not surprising that the kinetic maintenance of the convex hull of a set of n moving points in the plane was already studied by Basch et al. [8]. They designed a KDS that needs to be updated $O(n^{2+\epsilon})$ times, assuming the trajectories of the points are algebraic curves described by bounded degree polynomials. Later Agarwal et al. [6] proved that the number of events is in fact $(n\lambda_s(n))$ where s is the number of times any fixed triple of points can become collinear and $\lambda_s(n)$ is the maximum length of Davenport-Schinzel sequences of order s on n symbols. At each event, the KDS can be updated in $O(\log^2 n)$ time.

In some applications it may be necessary to maintain the attribute of interest explicitly. If one uses a KDS for collision detection, for instance, any external event—a collision in this case—must be reported. In such cases, the number of changes to the attribute is a lower bound on the number of events to be processed. Since the convex hull of n linearly moving points can change $\Omega(n^2)$ times [6], this means that any KDS that maintains an explicit representation of the convex hull must process $\Omega(n^2)$ events in the worst case. Hence, the convex-hull KDS of Basch et al. [8], which indeed maintains the convex hull explicitly, is close to optimal in the worst case.

In other applications, however, explicitly maintaining the attribute at all times may not be necessary; the attribute is only needed at certain times. This is for instance the case when a KDS is used as an auxiliary structure in another KDS. The auxiliary KDS is then used to update the main KDS efficiently when a certificate of the main KDS fails. In this case, even though the main KDS may have to be maintained explicitly, the attribute maintained by the auxiliary KDS only needs to be available at certain times. This leads us to view a KDS as a query structure: we want to maintain a set S of moving objects in such a way that we can reconstruct the attribute of interest efficiently whenever this is called for. This makes it possible to reduce the maintenance cost, as it is no longer necessary to update the KDS whenever the attribute changes. On the other hand, a reduction in maintenance cost will have an impact on the query time, that is, the time needed to reconstruct the attribute. Thus there is a trade-off between maintenance cost and query time, somewhat similar to storage versus query time trade-offs for e.g. range-searching data structures. The main goal of our paper is to study such trade-offs for kinetic convex hulls.

Our results. As just noted, our main interest lies in trade-offs between the maintenance cost of a kinetic convex-hull structure and the time to reconstruct the convex hull at any given time. To this end, we first study the simpler *kinetic sorting problem*: maintain a KDS on a set of n points moving on the real line such that at any time we can quickly reconstruct a sorted list of the points. We prove in Section 2 that already for the kinetic sorting problem one cannot get good trade-offs: even for linear motions, the worst-case maintenance cost is $\Omega(n^2)$ if one wants to be able to do the reconstruction in $o(n)$ time. Note that with $\Omega(n^2)$ maintenance cost, we can explicitly maintain the sorted list at all times, so that the reconstruction cost is zero. Thus interesting trade-offs are only possible in a very limited range of the spectrum, namely for reconstruction costs between $\Omega(n)$ and $O(n \log n)$. For this range we also prove lower bounds: we roughly show that one needs $\Omega(n^2/m)$ maintenance cost if one wants to achieve $o(n \log m)$ reconstruction cost, for any m with $2 \leqslant m \leqslant n$. (See Section 2.1 for a definition of our lower-bound model.) We also give a matching upper bound.

The negative results on the kinetic sorting problem make it quite unlikely that one can obtain good trade-offs for the kinetic convex-hull problem. (The results do not give a formal proof of this fact because the comparison-based model we use for the 1D sorting problem does not apply in 2D.) However, we will show that it is possible to get a good trade-off between maintenance and reconstruction time when the number of points on the convex hull is small: for any Q with $1 \leqslant Q \leqslant n$ and any $\epsilon > 0$ there is a KDS that processes $O(n^{2+\epsilon}/Q^{1+1/\delta})$ events such that one can reconstruct the convex hull of S in $O(hQ \log n)$ time, where δ is the maximum degree of the polynomials describing the motions of the points and h is the number of vertices of the convex hull. We obtain this result by giving a KDS for *gift-wrapping queries*: given a point q and a line ℓ through q such that all points from S lie on the same side of ℓ, report the point $p_i \in S$ that is hit first when ℓ is rotated (in counterclockwise direction, say) around q. Our KDS for this problem has $O(Q \log n)$ query time and it processes $O(n^{2+\epsilon}/Q^{1+1/\delta})$ events. For linear motions, this bound is very close to the lower bounds de Berg [9] proved for the kinetic dictionary problem—see below—which seems an

easier problem. Our KDS can also answer *extreme-point queries* (given a query direction $\vec{d}$, what is the point from S that is extreme in direction $\vec{d}$?) and *convex-hull containment queries* (given a query point q, is q inside the current convex hull?).

Related work. Some existing KDSs—the kinetic variants of various range-searching data structures [1–3,5,13,15, 16,19], for instance—do not maintain a uniquely defined attribute such as the convex hull, but they maintain a query data structure. In this setting the KDS is, of course, a query structure as well. Our setting is different because we are studying the maintenance of a single, uniquely defined, attribute such as the convex hull. This is somewhat similar to the papers by Guibas et al. [12] and by Hershberger and Suri [13], who study the kinetic maintenance of the connectivity of moving regions in the plane. Their structures can answer queries of the form: "Are regions A and B in the same connected component of the union of the regions?" However, their structure is updated whenever the connectivity changes—they do not allow for trade-offs between the number of events and the query time. Moreover, their goal is not to be able to quickly reconstruct the entire connectivity information at any given time. Thus their KDS is essentially a kinetic version of a structure for connectivity queries, rather than a kinetic query structure for reconstructing a unique attribute.

One of the main results of our paper is a lower bound on the trade-offs between reconstruction time and maintenance cost for the kinetic sorting problem. Lower bounds for trade-offs between query time and maintenance cost were also given by de Berg [9], but he studied the kinetic dictionary problem, where one wants to maintain a dictionary on a set S of n points moving on the real line. He showed that any kinetic dictionary with worst-case query time $O(Q)$ must have a worst-case total maintenance cost of $\Omega(n^2/Q^2)$, even if the points move linearly. As already remarked, the upper bounds we obtain for gift-wrapping and convex-hull containment queries on moving points in the plane—these problems seem at least as hard as the kinetic dictionary problem—almost match these bounds for linear motions.

A recent paper by Agarwal et al. [2] is closely related to the second part of our paper, where we describe a KDS for convex-hull containment queries. They describe, besides data structures for various range searching and proximity queries on moving points, a structure for convex-hull containment queries on moving points. Their structure is more powerful since it can answer queries about past or future convex hulls. On the other hand, they only deal with linear motions. Their structure uses $O(n^{2+\varepsilon}/Q^2)$ storage to obtain a query time of $O(Q \text{ polylog } n)$. Since the KDS is precomputed for the complete motions, there are no events. Note that the number of events we process for linear motions is the same as the amount of storage used by their structure. This means we can also obtain their result (namely the ability to answer queries in the past and future as well, at the expense of extra storage): during preprocessing, do a complete simulation based on the motions (that are assumed to be given) and record the changes to the KDS using standard persistency methods.

2. The kinetic sorting problem

Let $S = \{x_1, \ldots, x_n\}$ be a set of n point objects[3] moving continuously on the real line. In other words, the value of x_i is a continuous function of time, which we denote by $x_i(t)$. We define $S(t) = \{x_1(t), \ldots, x_n(t)\}$. For simplicity, we write S and x_i instead of $S(t)$ and $x_i(t)$, respectively, provided that no confusion arises. The kinetic sorting problem asks to maintain a structure on S such that at any given time t we can quickly generate a sorted list for $S(t)$. We call such a structure a *sorting KDS*.

We focus on trade-offs between the sorting cost and the maintenance cost: what is the worst-case maintenance cost if we want to guarantee a sorting cost of $O(Q)$, where Q is some parameter, under the assumption that the point objects follow trajectories that can be described by bounded degree polynomials. (In fact, for our lower bounds we will only use linear motions, whereas for our upper bounds we only need the restriction that any pair of points swaps $O(1)$ times.)

2.1. The lower-bound model

We shall prove our lower bounds for the kinetic sorting problem in the comparison-graph model introduced by de Berg [9], which is defined as follows. A *comparison graph* for a set S of numbers is defined as a directed graph

[3] We use "point objects" (or sometimes just "objects") for the points in S to distinguish them from other points that play a role in our proofs.

$\mathcal{G}(S, A)$ such that if $(x_i, x_j) \in A$, then $x_i < x_j$. The reverse is not true: the fact that $x_i < x_j$ does not mean there must be an arc in $\mathcal{G}$. The idea is that the comparison graph represents the ordering information encoded in a sorting KDS on the set S: if $(x_i, x_j) \in A$, then the fact that $x_i < x_j$ can be derived from the information stored in the KDS, without doing any additional comparisons.

Maintenance cost. The operations we allow on the comparison graph are insertions and deletions of arcs. For the maintenance cost, we only charge the algorithm for insertions of arcs; deletions are free. Note that by doing a single comparison we can sometimes obtain a lot of ordering information by transitivity. Therefore we only charge the algorithm for a new arc in the transitive reduction of the graph, that is, a new arc that is not implied by transitivity. Following de Berg [9], we therefore define the maintenance cost as the total number of such non-redundant arcs ever inserted into the comparison graph, either at initialization or during maintenance operations.

We say that the arc $(x_i, x_j) \in A$ *fails* at time t if $x_i(t) = x_j(t)$. The non-redundant arcs in the comparison graph essentially act as certificates, and their failures trigger events at which the KDS needs to be updated.

Query cost. A query at time t asks to construct a sorted list on the points in the current set S (that is, $S(t)$). We shall consider two different measures for the query cost.

The comparison-graph sorting model: The first measure is in a very weak model, where we only charge for the minimum number of comparisons needed to obtain a sorted list, assuming we have an oracle at our disposal telling us exactly which comparisons to do. This is similar to the query cost used by de Berg when he proved lower bounds for the kinetic dictionary. For the sorting problem this simply means that the query cost is equal to the number of pairs $x_i, x_j \in S$ that are adjacent in the ordering and for which there is no arc in the comparison graph.

The algebraic decision-tree model: In this model we also count the number of comparisons needed to sort the set S, but this time we do not have an oracle telling us which comparisons to do. We shall use the following basic fact: Suppose the number of different orderings of S that are compatible with the comparison graph at some given time is N. Then the cost to sort S in the algebraic decision-tree model is at least $\log N$.

2.2. A lower bound in the comparison-graph sorting model

The point objects in our lower-bound instance will move with constant (but different) velocities on the real line. Hence, if we view the line on which the point objects move as the x-axis and time as the t-axis, then the trajectories of the point objects are straight lines in the tx-plane. We use ξ_i to denote the line in the tx-plane that is the trajectory of x_i. It is somewhat easier to describe the lower-bound instance in the dual plane. We shall call the two axes in the dual plane the u-axis and the v-axis. We use the standard duality transform [10], where a line $\xi : x = at + b$ in the tx-plane is mapped to the point $\xi^* : (a, -b)$ in the dual plane, and a point $p : (a, b)$ in the primal plane is mapped to the line $p^* : v = au - b$ in the dual plane.

Now let $p_1, \ldots, p_n$ be the vertices of a regular n-gon in the dual plane that is oriented such that the diagonal $p_{l-1}p_{l+1}$ connecting the two neighbors of the leftmost vertex p_l is almost parallel to the v-axis and has negative slope—see Fig. 1. The trajectories $\xi_1, \ldots, \xi_n$ in our lower-bound instance are the primals of the vertices p_i, that is, $\xi_i^* = p_i$. In the remainder of this section we will prove a lower bound on the maintenance cost of any comparison graph for this instance whose sorting cost (in the comparison-graph sorting model) is bounded by Q, where Q is a parameter with $0 \leqslant Q < n$.

For any pair of vertices p_i, p_j, let ℓ_{ij} denote the line passing through p_i and p_j. Since the p_i are the vertices of a regular n-gon, the lines ℓ_{ij} have only n distinct slopes. Indeed, because the polygon is regular, the slope of edge $p_i p_{i+1}$ is the same as the slope of the diagonals $p_{i-1}p_{i+2}$, $p_{i-2}p_{i+3}$, etc. Note that ℓ_{ij} corresponds to the intersection of ξ_i and ξ_j in the tx-plane, with the slope of ℓ_{ij} being equal to t-coordinate of the intersection. This implies that the intersection points of the trajectories in the tx-plane have only n distinct t-values. Let $t_1, \ldots, t_n$ be the sorted sequence of these t-values. The times $t_1, \ldots, t_n$ define $n + 1$ open time intervals $(-\infty, t_1), (t_1, t_2), \ldots, (t_n, +\infty)$. Since no two trajectories intersect inside any of these intervals, the order of the point objects is the same throughout any interval. We say that x_i is *directly below* x_j in such an interval if $x_i(t) < x_j(t)$ for times t in the interval and there is no other point object x_k in between them in that interval. Furthermore, we call a vertex p_i a *lower vertex* if it lies on the lower part of the boundary of the n-gon, and we call p_i an *upper vertex* if it lies on the upper part of the boundary of the n-gon; the leftmost and rightmost vertices are neither upper nor lower vertices.

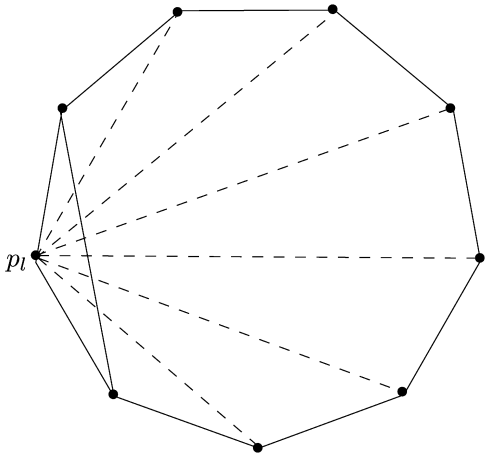

Fig. 1. The orientation of the regular n-gon.

Lemma 1.

(i) *The object x_i is directly below each other point object x_j in at least one time interval and at most three time intervals.*
(ii) *There are $\lceil n/2 \rceil$ objects x_i such that x_i is directly below each other object x_j in exactly one time interval.*

Proof.

(i) Consider two objects x_i, x_j. Since ξ_i and ξ_j intersect, there is at least one interval—just before or just after the intersection—where x_i is directly below x_j. Since there are $n+1$ intervals, and x_i is below each of the $n-1$ other objects in at least one interval, x_i can be directly below x_j in at most three time intervals.
(ii) First we show that if p_i is a lower vertex or leftmost vertex, x_i is directly below each other object x_j in exactly one time interval. Since there are $n+1$ intervals, and x_i is below each of the $n-1$ other objects in at least one interval, we need to prove that x_i is above all other objects in the two remaining intervals. To prove this, we note that ξ_i appears on the upper envelope of the trajectories in the tx-plane, since p_i is a lower or leftmost vertex. Let p_i be a lower vertex. Assume that ξ_i appears on the upper hull at time t when it intersects ξ_j, and assume it disappears at time t' when it intersects $\xi_{j'}$. Then ξ_j and $\xi_{j'}$ must cross between times t and t', showing that ξ_i is on the upper envelope during two time intervals.
Now consider the leftmost vertex p_l. This vertex is on the upper envelope at time $t = -\infty$. Because of the orientation of the n-gon, the slope of the diagonal connecting the two neighbors of p_l is smaller than the slope of any diagonal (or edge) incident to p_l—see Fig. 1. This means that the two objects that are initially below ξ_l^* intersect before ξ_l^* disappears from the envelope, which proves x_l is also above all other objects in the two remaining intervals.
When n is even, the number of vertices that are a lower or leftmost vertex is $n/2$, and we are done. For odd n, we will argue that the object x_r corresponding to the rightmost vertex is also directly below each other object exactly once.
To show this, we will argue that ξ_r appears on the upper envelope during two time intervals. Indeed, there are $n+1$ time intervals and the $\lfloor n/2 \rfloor$ leftmost and lower vertices correspond to trajectories appearing twice. Hence there are $n+1-\lfloor n/2 \rfloor = 2$ time intervals where some other trajectory must appear on the upper envelope. This must be ξ_r, because the trajectories of upper vertices will not show up on the upper envelope. □

We can now prove the lower bound. Suppose that we have a comparison graph on the point objects whose sorting cost is Q during each of the time intervals defined above. This implies that during each such time interval, there must be at least $n-Q-1$ arcs (x_i, x_j) in the comparison graph such that x_i is directly below x_j, because each of the $n-1$ adjacent pairs must have an arc, and we are allowed to add only Q arcs to answer the query. In total, $(n+1)(n-Q-1)$

arcs are needed over all $n+1$ time intervals. Some arcs, however, can be used in more than one interval. For x_i, let k_i be the number of arcs of the form (x_i, x_j) that are used. For any of the $\lceil n/2 \rceil$ objects x_i for which case (ii) of Lemma 1 applies, all these arcs are distinct. For the remaining $\lfloor n/2 \rfloor$ objects case (i) applies and so at least $k_i - 2$ arcs are distinct. Hence, the total number of arcs inserted over time is at least $(n+1)(n-Q-1) - 2\lfloor n/2 \rfloor \geqslant n(n-Q-2)$. We get the following theorem.

Theorem 2. *For any $n \geqslant 1$, there is an instance of n point objects moving with constant velocities on the real line, such that any comparison graph whose worst-case sorting cost in the comparison-graph cost model is Q must have maintenance cost at least $n(n-Q-2)$, for any parameter Q with $0 \leqslant Q < n$.*

2.3. A lower bound in the algebraic decision-tree model

In the previous section we gave a lower bound for the maintenance cost for a given sorting cost Q in comparison-graph sorting model. Obviously, this is also a lower bound for the algebraic decision-tree model. Hence, the results of the previous section imply that for any sorting cost $Q = o(n)$ in the algebraic decision-tree model, the worst-case maintenance cost is $\Omega(n^2)$. Since with $O(n^2)$ maintenance cost we can process all swaps—assuming the trajectories are bounded-degree algebraic, so that any pair swaps at most $O(1)$ times—this bound is tight: with $O(n^2)$ maintenance cost we can achieve sorting cost zero. What remains is to investigate the range where the sorting cost is $\Omega(n)$ and $o(n \log m)$, where $1 < m \leqslant n$.

Recall that the sorting cost of a given comparison graph in the algebraic decision-tree model is at least $\log N$, where N is the number of different orderings that are compatible with the comparison graph. We use this to prove the following lemma.

Lemma 3. *There is a positive constant c such that if the sorting cost of a comparison graph is at most $cn \log m$, then there is a path in the comparison graph whose length is at least $n/m^{1/3}$.*

Proof. Let k be the length of the longest path in the comparison graph. We define the *level* of the point object x_i as the length of the longest path to x_i in the comparison graph. Let n_j be the number of objects at level j. Since the order of the objects in the same level is not determined by the information in the comparison graph, the number of permutations compatible with the comparison graph is at least $n_0!n_1!\cdots n_k!$. Note that $\sum_{i=0}^{k} n_i = n$, so $n_0!n_1!\cdots n_k!$ is minimized when the n_i's are all equal to $\lceil n/(k+1) \rceil$ or $\lfloor n/(k+1) \rfloor$. Hence

$$n_0!n_1!\cdots n_k! \geqslant \left(\left\lfloor \frac{n}{k+1} \right\rfloor ! \right)^{k+1},$$

and the sorting cost in the algebraic decision-tree model is at least $\log((\lfloor n/(k+1) \rfloor !)^{k+1})$, which is at least $c_1 n \log(n/k)$ for some constant c_1. It follows that we must have $c_1 \log(n/k) \leqslant c \log m$, which implies that $n/k \leqslant m^{c/c_1}$. So for $c = c_1/3$ we have $k \geqslant n/m^{1/3}$, as claimed. □

Next we describe the lower bound construction. As before, it will be convenient to describe the construction in the dual plane. To this end, let $G_a := \{0, 1, \ldots, a-1\}^2$ be the $a \times a$ grid. The trajectories of the point objects in our lower-bound instance will be straight lines in the tx-plane, such that the duals of these lines are the grid points of $G_{\sqrt{n}}$. (We assume for simplicity that n is a square number.) Before we proceed, we need the following lemma.

Lemma 4. *Let $p = (p_x, p_y)$ be a grid point of $G_{\sqrt{n}}$ and $p_x, p_y \leqslant a$, where $a \leqslant \sqrt{n}/2$. Let ℓ_p be the line through the origin and p. The number of different lines passing through at least one point of $G_{\sqrt{n}}$ and being parallel to ℓ_p is at most $4a\sqrt{n}$.*

Proof. Let B be the smallest box containing $G_{\sqrt{n}}$ that has one edge (in fact, two) parallel to ℓ_p. Thus B is a bounding box for $G_{\sqrt{n}}$ whose orientation is defined by ℓ_p. Besides the points from $G_{\sqrt{n}}$, the box B will contain more points with integer coordinates; in the remainder of this proof we will call these points grid points as well. The number of grid points inside B is at most $2n$. Moreover, because $p_x, p_y \leqslant a$, any line passing through a grid point and being

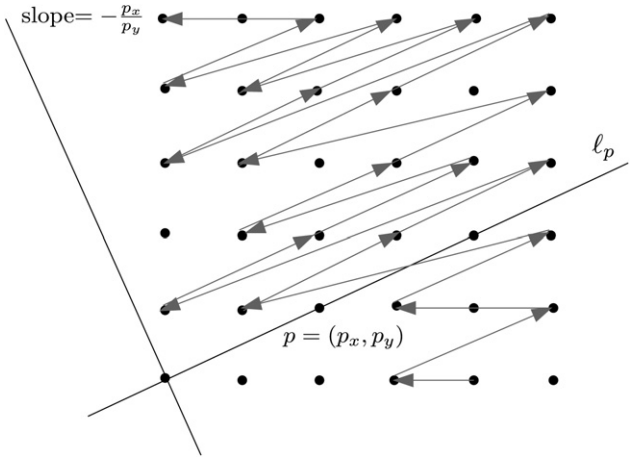

Fig. 2. A long path uses many arcs parallel to ℓ_p.

parallel to ℓ_p contains at least $\lfloor\sqrt{n}/a\rfloor \geqslant \sqrt{n}/(2a)$ grid points inside B. Since two distinct lines parallel to ℓ_p cannot have any grid points in common, this implies that the number of such lines containing at least one grid point is at most

$$\frac{2n}{\sqrt{n}/(2a)} = 4a\sqrt{n}. \qquad \square$$

We are now ready to prove the main result of this section.

Theorem 5. *For any $n \geqslant 2$, there are positive constants c and c' such that there is an instance of n point objects moving with constant velocities on the real line such that, for any m with $1 < m \leqslant n$, any comparison graph whose worst-case sorting cost in the algebraic decision-tree model is $Q \leqslant cn\log m$, must have maintenance cost at least $c'n^2/m$.*

Proof. As mentioned above, the trajectories ξ_i of the point objects x_i that constitute our lower-bound instance will be straight lines in the tx-plane, whose dual points ξ_i^* form the grid $G_{\sqrt{n}}$.

Let $a := \sqrt{n}/(8m^{1/3})$ and c be the constant of Lemma 3. Consider the comparison graph at some time $s + \varepsilon$ with $s = p_y/p_x$, where $p_x, p_y \leqslant a$ and $\varepsilon > 0$ is sufficiently small. Suppose the sorting cost at time s is at most $cn\log m$. Then by Lemma 3 there must be a path in the comparison graph of length at least $n/m^{1/3}$. We claim (and will prove below) that at least half of the arcs in this path are between point objects x_i, x_j such that ξ_i^* and ξ_j^* lie on a common line of slope s—see Fig. 2. The number of distinct values for s is equal to the number of pairs (p_x, p_y) where p_x and p_y are integer numbers between 0 and $a-1$ (including 0 and $a-1$) and $GCD(p_x, p_y) = 1$. Because of symmetry, we count the number of pairs (p_x, p_y) with the property $p_x \leqslant p_y$. For a nonnegative integer i, let $\varphi(i)$ be the number of nonnegative integers that are less than i and relatively prime to i. Then the number of pairs (p_x, p_y) with the desired properties is $\sum_{i=1}^{a-1}\varphi(i)$. It is known [20] that this summation is $\Theta(a^2)$. Then, the total number of arcs needed over all times of the form $p_x/p_y + \varepsilon$ with $p_x, p_y \leqslant a$ is at least

$$n/(2m^{1/3}) \cdot \Theta(a^2) \geqslant c'n^2/m \quad \text{for some constant } c',$$

which proves the theorem.

It remains to prove the claim that at least half of the arcs in the path are between point objects x_i, x_j such that ξ_i^* and ξ_j^* lie on a common line of slope s. Note that the sorted order of the point objects $x_i(s+\varepsilon)$ corresponds to the sorted order of the orthogonal projections of the points ξ_i^* onto a line with slope $-1/(s+\varepsilon)$. If $\varepsilon > 0$ is sufficiently small, then the projections of all the points lying on a common line of slope s will be adjacent in this order. Let's group the point objects x_i into subsets such that any two point objects x_i, x_j for which ξ_i^* and ξ_j^* lie on a common line of slope s are in the same subset. Then, at time $s+\varepsilon$, any path in the comparison graph can enter and leave a subset at

most once. By Lemma 4 the number of subsets is at most $4a\sqrt{n}$. Hence, the number of arcs connecting point objects in the same subset is at least

$$n/m^{1/3} - 4a\sqrt{n} = n/m^{1/3} - 4\big(\sqrt{n}/(8m^{1/3})\big)\sqrt{n} = n/(2m^{1/3}),$$

as claimed. □

2.4. Upper bounds for kinetic sorting

The following theorem shows the bounds in Theorem 5 are tight.

Theorem 6. *Let S be a set of n point objects moving on the line, such that any pair of points swaps $O(1)$ times. For any m with $1 < m \leqslant n$, there is a data structure whose maintenance cost is $O(n^2/m)$ such that at any time a sorted list of the points in S can be constructed in $O(n \log m)$ time.*

Proof. Partition the set S into m subsets of size at most n/m in an arbitrary manner. For each subset, maintain a sorted array of all its points. When two point objects in the same subset swap, the array can be updated in $O(1)$ time. Since each pair of objects changes order $O(1)$ times, the maintenance cost of each subset is $O(n^2/m^2)$. Since there are m subsets, the total maintenance cost is $O(n^2/m)$. To generate a sorted list of all the point objects, we have to merge the m sorted arrays, which can be done in $O(n \log m)$ time. □

3. Gift-wrapping and convex-hull containment queries

Let $S = \{p_1, \ldots, p_n\}$ be a set of point objects moving in the plane such that the position of p_i at time t is $(x_i(t), y_i(t))$, where x_i and y_i are polynomials of degree at most δ. Recall that a *gift-wrapping query* is defined as follows: given a point q and a line ℓ through q such that all points from S lie on the same side of ℓ, report the point object $p_i \in S$ that is hit first when ℓ is rotated (in counterclockwise direction, say) around q. We call a KDS for such queries a *gift-wrapping KDS*. We want to find a gift-wrapping KDS of near-linear size with good trade-offs between the maintenance cost and the query time. We also want to answer *extreme-point queries* (given a query direction $\vec{d}$, what is the point from S that is extreme in direction $\vec{d}$?) and *convex-hull containment queries* (given a query point q, is q inside the current convex hull?).

One easy solution is the following: partition the set S into Q subsets of roughly size n/Q and maintain each subset using the kinetic convex-hull structure of Basch et al. [8]. Since we can answer a gift-wrapping query on each subset in $O(\log n)$ time (if we have the convex hull in a sorted array or balanced search tree), we can answer gift-wrapping queries on S in $O(Q \log n)$ time. Extreme-point queries and convex-hull containment queries can also be answered with this structure. The total maintenance cost will for this KDS would be $Q \cdot (n/Q)^{2+\epsilon} = O(n^{2+\epsilon}/Q)$. Next we describe a KDS that can answer all three types of queries as well, and is more efficient than the easy solution described above.

The data structure. Consider the following transformation on (the trajectories of) the point objects in S: the point object $p_i(t) = (x_i(t), y_i(t))$, where $x_i(t) = x_{i,\delta}t^\delta + \cdots + x_{i,0}$ and $y_i(t) = y_{i,\delta}t^\delta + \cdots + y_{i,0}$, is mapped to the point $p_i^* = (x_{i,\delta}, \ldots, x_{i,0}, y_{i,\delta}, \ldots, y_{i,0})$ in $2(\delta+1)$-dimensional space.

Lemma 7. *If the point object $p_i(t)$ is more extreme than $p_j(t)$ in direction $\vec{d} = (d_x, d_y)$, then p_i^* is more extreme than p_j^* in direction $\vec{d}^* = (t^\delta d_x, \ldots, t d_x, d_x, t^\delta d_y, \ldots, t d_y, d_y)$.*

Proof. If $p_i(t)$ is more extreme than $p_j(t)$ in direction $\vec{d} = (d_x, d_y)$, then $p_i(t) \cdot \vec{d} > p_j(t) \cdot \vec{d}$. Plugging in the polynomials defining the coordinates of $p_i(t)$ we see that this is equivalent to $p_i^* \cdot \vec{d}^* > p_j^* \cdot \vec{d}^*$. Hence, p_i^* is more extreme than p_j^* in direction $\vec{d}^*$. □

Our gift-wrapping KDS is a combination of the data structure for half-space emptiness queries (in $2(\delta+1)$-dimensional space) as described by Matoušek and Schwarzkopf [18], and the kinetic convex-hull structure (in the plane) of Basch et al. [8]. It is defined recursively, as follows.

Let $S_v \subset S$ be the subset of points for which we are constructing the KDS. Initially, $S_v = S$.

- If $|S_v| \leqslant n/Q^{1+1/\delta}$, where n is the number of points in the original set S, then S_v is stored in the kinetic convex-hull structure of Basch et al. [8].
- Otherwise we transform (the trajectories of) the points in S_v to obtain a static set S_v^* of points in $2(\delta+1)$-dimensional space as described above, and we proceed as Matoušek and Schwarzkopf [18]: We construct a simplicial partition Ψ_v for S_v^*—a partitioning of S_v^* into $O(r)$ subsets $S_{v,i}^*$ for some suitably large constant r, each of size between n/r and $n/2r$, and for each subset $S_{v,i}^*$ a simplex containing it—using Matoušek's partition theorem for shallow hyperplanes [17]. The simplicial partitioning Ψ_v has the following property: any hyperplane h for which one of its half-spaces has less than n/r points from S_v^* crosses $O(r^{1-1/\lfloor d/2\rfloor}+\log r)$ simplices of Ψ_v. We also construct a $(1/r)$-net R_v^* of size $O(r\log r)$ for S_v^*, that is, a subset of S_v^* such that any halfspace containing at least n/r points of S_v^* must contain at least one point of R_v^*.
  The structure now consists of a root node where we store the simplices of Ψ_v and the $(1/r)$-net R_v^*. The root has a subtree for each of the subsets $S_{v,i}^*$ (or rather, the set $S_{v,i}$ of their pre-images), which is constructed recursively.

Gift-wrapping queries. A gift-wrapping query with a line ℓ rotating around a point q can be answered as follows. Suppose we are at some node v of the structure. If $|S_v| \leqslant n/Q^{1+1/\delta}$, then we have the convex hull of S_v explicitly available, so we can answer the query in $O(\log|S_v|)$ time. Otherwise, we find the point $p_i \in R_v$ (the set of point objects that are the pre-images of the points in R_v^*) hit first by ℓ, in a brute-force manner in $O(|R_v|) = O(r\log r)$ time. Let $\ell_q(p_i)$ be the line through q and p_i. Note that all points from R_v lie to the same side of, or on, $\ell_q(p_i)$. Let $\vec{d}$ be the vector orthogonal to $\ell_q(p_i)$ and pointing in the direction where there are no points from R_v. Then the answer to the query must either be the point p_i, or it must be a point $p_j \in S_v \setminus R_v$ that is more extreme than p_i in the direction $\vec{d}$. Transform $\vec{d}$ into a vector $\vec{d}^*$ in $2(\delta+1)$-dimensional space, as in Lemma 7, let h^* be the hyperplane through p_i^* and orthogonal to $\vec{d}^*$, and let $(h^*)^+$ be the half-space defined by h^* and the vector $\vec{d}^*$. By Lemma 7, any point p_j that is more extreme than p_i in direction $\vec{d}$ is mapped to a point p_j^* that lies in $(h^*)^+$. Moreover, none of the points in R_v^* lie in $(h^*)^+$, which means that $(h^*)^+$ contains less than n/r points. It follows that no simplex of Ψ_v can lie completely inside $(h^*)^+$. Hence, the query can be answered by recursing only into the subtrees corresponding to intersected simplices, and selecting from the answers found the first point hit.

Extreme-point queries. Extreme-point queries can be answered in a similar way: if we are at a node v with $|S_v| \leqslant n/Q^{1+1/\delta}$, answer the query using the convex hull of S_v. Otherwise, find the point $p_i \in R_v$ that is extreme in the direction $\vec{d}$, and recurse into subtree corresponding to simplices of Ψ_v that are intersected by h^*, where h^* is the hyperplane through p_i^* and orthogonal to $\vec{d}^*$.

Convex-hull containment queries. Let $CH(S)$ denote the convex hull of a set S. The query point q lies outside $CH(S)$ if only if there are two half-lines with origin q and tangent to $CH(S)$. Our algorithm is recursive. Suppose we are at some node v of the structure. The algorithm returns true when $q \in CH(S_v)$ and it returns two tangent half-lines for S_v otherwise.

If $|S_v| \leqslant n/Q^{1+1/\delta}$, we have $CH(S_v)$ available. We test whether $q \in CH(S_v)$ and if so return true. Otherwise we return two tangent half-lines for $CH(S_v)$. This takes $O(\log|S_v|)$ time.

Now, assume v is an internal node. If $q \in CH(R_v)$, then $q \in CH(S_v)$ and we return true. Otherwise, two points $p_i, p_j \in R_v$ are computed such that the lines ℓ_{p_i} and ℓ_{p_j} passing through q and p_i resp. p_j are tangent to $CH(R_v)$. Let $\vec{d_{p_i}}$ ($\vec{d_{p_j}}$) be the vector orthogonal to ℓ_{p_i} (ℓ_{p_j}) and pointing in the direction where there are no points from R_v. Let $h_{p_i}^*$ ($h_{p_j}^*$) be the hyperplane through p_i^* (p_j^*) and orthogonal to $\vec{d_{p_i}}$ ($\vec{d_{p_j}}$). Using the same reasoning as for gift-wrapping queries, we can argue that we only have to recurse into subtrees of simplices intersected by $h_{p_i}^*$ or $h_{p_j}^*$. If one of these calls returns true, we also return true. Otherwise we collect all the tangent half-lines. If there is no line through q such that all half-lines lie to the same side of the line, we return true. Otherwise, from these we can easily select the two half-lines that are tangent to $CH(S_v)$ and return them.

Theorem 8. *Let $S = \{p_1, \ldots, p_n\}$ be a set of moving point objects in the plane such that the position of $p_i(t) = (x_i(t), y_i(t))$, where x_i and y_i are polynomials with degree at most δ. For any Q with $1 \leqslant Q \leqslant n$ and any $\epsilon > 0$ there is a KDS that handles $O(n^{2+\epsilon}/Q^{1+1/\delta})$ events such that gift-wrapping queries, extreme-point queries, and convex-hull containment queries can be answered in $O(Q\log n)$ time. The KDS uses $O(n\log n)$ storage, and events can be handled in $O(\log^2 n)$ time.*

Proof. The partition-tree part of our KDS is static—it stores the trajectories rather than the current positions of the points—so events only occur in the kinetic convex-hull structures. There are $Q^{1+1/\delta}$ such structures, each of them storing at most $n/Q^{1+1/\delta}$ point objects and processing $O((n/Q^{1+1/\delta})^{2+\epsilon})$ events [8]. In total, this gives $O(n^{2+\epsilon}/Q^{1+1/\delta})$ events. The bounds on the storage and the time needed to handle an event follow directly from the bounds on the kinetic convex-hull structure [8].

Next we bound the time for a gift-wrapping query; the analysis for extreme-point queries and for convex-hull containment queries is similar. $T(m)$, the query time on a subtree storing m points, satisfies the following recurrence:

$$T(m) = \begin{cases} O(\log m) & \text{if } m \leqslant \frac{n}{Q^{1+1/\delta}}, \\ O(r \log r) + O(r^{1-1/(\delta+1)} + \log r) \cdot T(2m/r) & \text{otherwise.} \end{cases}$$

For any ϵ, we can choose r sufficiently large such that the solution of the recurrence is $O(Q^{1+\epsilon} \log m)$. The factor Q^{ϵ} in the query time can be avoided by replacing Q by $Q^{1-\varepsilon}$; this gives an extra factor $Q^{\epsilon(1+1/\delta)})$ in the number of events, which is swallowed by the n^{ϵ} factor that we already have in the number of events. □

Remark. Instead of switching to the kinetic convex hull structure of Basch et al. when the number of points becomes small, we could also dualize the points in S_v^* and switch to a structure based on cuttings in $2(\delta+1)$-dimensional space. This would lead to a structure with the same query time and no events to be processed, but with a much higher storage cost.

Using the gift-wrapping KDS we can easily reconstruct the convex hull:

Corollary 9. *Let $S = \{p_1, \ldots, p_n\}$ be a set of moving point objects in the plane such that the position of $p_i(t) = (x_i(t), y_i(t))$, where x_i and y_i are polynomials with degree at most δ. For any Q with $1 \leqslant Q \leqslant n$ and any $\epsilon > 0$ there is a kinetic data structure that handles $O(n^{2+\epsilon}/Q^{1+1/\delta})$ events, such that we can reconstruct the convex hull of S at any time in $O(hQ \log n)$ time, where h is the number of vertices of the convex hull. The KDS uses $O(n \log n)$ storage, and each event can be handled in $O(\log^2 n)$ time.*

Proof. Maintain the gift-wrapping KDS of Theorem 8 on the points, and maintain a kinetic tournament tree [8] on the y-coordinates of the points. Using the kinetic tournament tree, we always have the lowest point of S available, which implies that we can reconstruct the convex hull by $O(h)$ gift-wrapping queries. The number of events in the kinetic tournament tree is $O(n \log n)$, which is subsumed by the number of events in the gift-wrapping KDS. □

Note that the structure can not easily handle flight plan updates–see also the discussion of the end of the conclusion.

4. Conclusions

We have studied trade-offs for the kinetic sorting problem, which is to maintain a KDS on a set of points moving on the real line such that one can quickly generate a sorted list of the points, at any given time. We have proved a lower bound for this problem showing the following: with a subquadratic maintenance cost one cannot obtain any significant speed-up on the time needed to generate the sorted list (compared to the trivial $O(n \log n)$ time), even for linear motions.

This negative result gives a strong indication that good trade-offs are not possible for a large number of geometric problems—Voronoi diagrams and Delaunay triangulations, for example, or convex hulls—as the sorting problem can often be reduced to such problems (this is not a formal proof, because our lower-bound model is not suitable for computing convex hulls or Voronoi diagrams). For the convex-hull problem, however, we have shown that good trade-offs are possible if the number of vertices of the convex hull is small. We obtained this result by developing a KDS for gift-wrapping queries, which is of independent interest. Our structure can also answer extreme-point queries and convex-hull containment queries. It would be interesting to see if we can develop a KDS with a similar performance for line-intersection queries: report the intersection points of a query line ℓ with the current convex hull.

Another open problem is to make the KDS less sensitive to changes in the motions of the points. In our structure, a change in motion means we have to delete the point (or rather, its trajectory) from the structure and reinsert the new

trajectory. Using the dynamic version of the structure of Matoušek and Schwarzkopf [18] this might be possible, but it would be nicer if we had a structure where no changes are needed (except for a recomputation of the failure times) when a point changes its motion. Note, however, that with a small change in the definition of our structure we can at least ensure that it will function correctly when a point changes its motion. All we have to do is add at every node v one point from each simplex in Ψ_v to the net R_v. With this change the KDS will always report the correct answer, even if we keep the wrong trajectories in the top part of our structure (we still have to update the kinetic convex-hull structures that we store at the "leaves" of our structures, of course, using the update algorithm of Basch et al.). Now we no longer have any guarantees on the query time, however.

References

[1] P.K. Agarwal, L. Arge, J. Erickson, Indexing moving points, in: Proc. ACM Sympos. Principles Database Syst., 2000, pp. 175–186.
[2] P.K. Agarwal, L. Arge, J. Erickson, H. Yu, Efficient tradeoff schemes in data structures for querying moving objects, in: Proc. 12th European Sympos. on Algorithms, 2004, pp. 4–15.
[3] P.K. Agarwal, L. Arge, J. Vahrenhold, Time responsive external data structures for moving points, in: Proc. 7th Workshop on Algorithms and Data Structures, 2001, pp. 50–61.
[4] P.K. Agarwal, J. Basch, M. de Berg, L.J. Guibas, J. Hershberger, Lower bounds for kinetic planar subdivisions, Discrete Comput. Geom. 24 (2000) 721–733.
[5] P.K. Agarwal, J. Gao, L.J. Guibas, Kinetic medians and kd-trees, in: Proc. 10th European Sympos. on Algorithms, 2002, pp. 5–16.
[6] P.K. Agarwal, L.J. Guibas, J. Hershberger, E. Veach, Maintaining the extent of a moving point set, Discrete Comput. Geom. 26 (2001) 353–374.
[7] P.K. Agarwal, S. Har-Peled, Maintaining approximate extent measures of moving points, in: 12th ACM-SIAM Symp. Discrete Algorithms, 2001, pp. 148–157.
[8] J. Basch, L.J. Guibas, J. Hershberger, Data structures for mobile data, J. Algorithms 31 (1999) 1–28.
[9] M. de Berg, Kinetic dictionaries: How to shoot a moving target, in: Proc. 11th European Sympos. on Algorithms, 2003, pp. 172–183.
[10] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer-Verlag, Berlin, 1997.
[11] L.J. Guibas, Kinetic data structure: a state of art report, in: Proc. 3rd Workshop Algorithmic Found. Robot., 1998, pp. 191–209.
[12] L.J. Guibas, J. Hershberger, S. Suri, L. Zhang, Kinetic connectivity for unit disks, Discrete Comput. Geom. 25 (2001) 591–610.
[13] J. Hershberger, S. Suri, Kinetic connectivity of rectangles, in: Proc. 15th ACM Sympos. Comput. Geom., 1999, pp. 237–246.
[14] D. Kirkpatrick, B. Speckmann, Kinetic maintenance of context-sensitive hierarchical representations of disjoint simple polygons, in: Proc. 18th ACM Sympos. Comput. Geom., 2002, pp. 179–188.
[15] G. Kollios, D. Gunopulos, V. Tsotras, Nearest neighbor queries in a mobile environment, in: Proc. Intl. Workshop on Spatiotemporal Database Managment, 1999, pp. 119–134.
[16] G. Kollios, D. Gunopulos, V. Tsotras, On indexing mobile objects, in: Proc. ACM Sympos. Principles Database Syst., 1999, pp. 261–272.
[17] J. Matoušek, Reporting points in halfspaces, Computational Geometry 2 (1992) 169–186.
[18] J. Matoušek, O. Schwarzkopf, On ray shooting in convex polytopes, Discrete Comput. Geom. 10 (1993) 215–232.
[19] S. Saltenis, C. Jensen, S. Leutenegger, M. Lopez, Indexing the position of continuously moving objects, in: Proc. SIGMOD Intl. Conf. Managment of Data, 2000, pp. 331–342.
[20] E.W. Weinstein, CRC Concise Encyclopedia of Mathematics, CRC Press, 1999.